1.Introduction

This project is composed of server and user nodes and a two-phase commit process as well as several other helper classes for better modularization. The main contents are about the communication protocol for two-phase commit between the single sever and other user nodes, then it also implements failure recovery mechanism and message retransmission mechanism.

2.Two-phase commit process and communication protocol

The design is implemented mainly based on two-phase commit with the corresponding communication protocol. This implementation uses TIMEOUT mechanism in both phases, which means if not all the votes are collected within a certain period, the server then took it as failed and broadcasts all the other notes abortion message. In the second phase, the server will resend the decision repeatedly if timeout. The user node also implements a blocking mechanism which means all the files involved cannot be proposed again till a global decision received to support concurrency. In both phases, the User Node does not check the TIMEOUT and only relies on the Server to detect the missed messages.

The projectLib class is the communication bridge between server and user nodes. There is a dedicated SerialMessage class which serialize the information of message to reduce the number of message transmissions. There are four types of Server message i.e. “Prepare”, “Commit”, “Abort” and “Done”, and three types of UserNode message, i,e., “YES”, “No”, “Ack”. Message is labeled with the filename and the message type. Server always keeps the source file numbers and their name to benefit user nodes that they don’t need to maintain these information themselves. The content of the collage is also stored and sent to the user node through message transmission, and this is used for decision. The file content is written to the disk after the agree decision is made.

3. Failure Handling and Recovery

This final implementation concludes: a: logging mechanism; b: TIMEOUT, to process runtime failures. The server is used to handle the lost message while usernode is implemented simply. In the first phase, a lost message is simply taken as a disagree by the server and in the second phase, the server will send the message again to confirm if all the usernodes already received the decision. The timeout for both phases are three seconds. To support the resending mechanism, the user node is implemented to be idempotent to ensure that if receiving the same decision repeatedly, everything will be consistent. Both server and user nodes continues to write log when they reach a checkpoint. To make it simple, we simply store the messages that can identify the checkpoints in the log, and in the time of recovery, we just reprocess the messages.

Whenever we send out a message that will change the state of the system:

- The starting of a commit

• Reached an agreement

• Actual commit / abort

• Done We store the message to the WAL file.

When a server or a user node reboot, it will check the log to know its recovery mode.

To be more specific, if there is not a <id>.wal file in the work space, then it starts in the Normal mode. If the there exists a <id>.wal file, its content is read into memory and each commit has all its messages sorted by time. One the server side, for each commit id,

• if we see a DONE message, then there is nothing we need to do.

• If we see a commit / abort message, then we need to resend the decision to all UserNodes.

• Otherwise the commit is still in the first phase, and we will send out abort messages to all UserNodes.

For the later two cases, we also need to rebuild the in-memory data structure for tracking and re-commit the file if needed.

For the user nodes, if the log contains a commit id that is not done, it simply replays the action (like holding locks and sending replies). Since the Server has the necessary global information, even if the UserNode sends messages out of order or obsolete, Server can handle it and respond to UserNode correctly

4. Use of Code

A set of sample images and test scripts are provided in test.tar. To use these, untar the file to produce a test directory with a set of subdirectories. Change your working directory to "test". Then, from this directory, run Project4, e.g.: **java Project4 15440 scripts/1-simple-commits.txt.** The Server will run in the Server directory, and up to 4 User Nodes in a, b, c, and d. Ensure that the committed composite images are generated in the Server directory, and that the corresponding sources are removed from the User Node directories. There are two additional scripts provided as well.

To run the test again, it is simply best to completely remove the test directory and recreate it from test.tar. This will ensure that all of the directories are in a clean state.